N92-16609

A CLIPS BASED PERSONAL COMPUTER HARDWARE DIAGNOSTIC SYSTEM

George M. Whitson

Computer Science Department
The University of Texas at Tyler
Tyler, Texas 75701

Abstract. Often the person designated to repair personal computers has little or no knowledge of how to repair a computer. This paper describes a simple expert system to aid these inexperienced repair people. The first component of the system leads the repair person through a number of simple system checks such as making sure all cables are tight and that the dip switches are set correctly. The second component of the system assists the repair person in evaluating the error codes generated by the computer. The final component the system applies a large knowledge base to attempt to identify the component (components) of the personal computer that is (are) malfunctioning. We have implemented and tested our design with a full system to diagnose problems for an IBM compatible system based on the 8088 chip. In our tests, the inexperienced repair people found the system very useful in diagnosing their hardware problems.

STATEMENT OF THE PROBLEM

Many universities and corporations have a large number of personal computers. Many of these personal computers have been purchased by departments that have little expertise in computer repair. When these computers need to be repaired someone in the department becomes the designated repair person. This person rarely has the expertise to do the repairs and wastes much time acquiring the expertise of a repair person. Since other computers of the same type are usually available it is easy to test to see if a component is faulty if one has the expertise. This is clearly an ideal situation for an expert system. The main purpose of the expert system is to apply the many rules that human computer experts remember to apply such as "is the floppy disk motor running" at exactly the correct time. A secondary purpose of the expert system is to apply a huge database of information about numeric error codes that are known for the system board and component boards in a meaningful fashion.

KNOWLEDGE ACQUISITION

As with all expert systems, the knowledge acquisition phase of this expert system was the most difficult part of its development. To begin the process, I discussed with several expert the overall design of the system and, after several sessions, we agreed on the

overall design presented in the next section. The experts provided lists of error codes and our expert system design was then implemented to give optimal organization of the rules for the error code analysis. The experts then designed a set of help screens that were driven by a knowledge base to lead the inexperienced user through a set of standard system checks to be sure that no obvious problems were causing the computer to malfunction. Finally, the experts were interviewed over a number of sessions to build the main knowledge base that was to assist the inexperienced user in identifying the exact component(s) that was (were) causing the computer to malfunction.

To enhance the ability of the Knowledge Engineers in obtaining the knowledge from the experts, I set up a room with test computers and a large set of boards that had been removed from other computers during repairs. In observing how the repair person discovered the bad boards or components, the rules were determined in relatively short order. After a prototype expert system was developed, a few rules were added and modified and, after our full system was tested, we added a few more.

In general, our technique consisted of the following steps:

(1) Design of the phases of the system.
(2) Quick discovery of most of the rules by using a special hardware lab.
(3) Modification of the knowledge base after the development of a prototype.
(4) Further modification of the system after the first full version of the system was used.

SYSTEM DESIGN

The expert system tool that we choose to use for our expert system was CLIPS. The simple structure of the shell and the ability to interface it to graphics and database packages makes it ideal for this type of expert system. After our initial interview of the experts, we decided to use the system architecture shown in Figure 1.

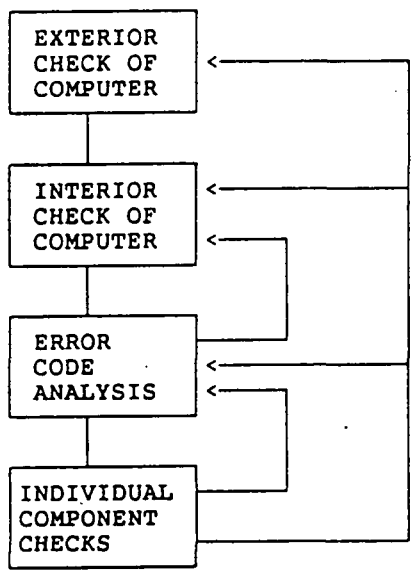

Figure 1.

The system has four major phases that, in general, will be executed in order, with the computer malfunction being detected at the earliest possible time. Each of the phases had its own rule base, with the most complex and interrelated rules being those of the fourth phase. As indicated in the architecture, some of the rules in a later phase could place the user back in an earlier phase.

INITIAL CHECK OF THE SYSTEM

This is an important component of the system. The knowledge base consists of a number of rules that present some help screens in an intelligent fashion and prompts the user to check for obvious system problems that are not related to a component failure. A typical help screen rule is given in Figure 2.

```
(defrule pc_power
  (or ?temp <- (power N)
      ?temp <- (power D)
  )
  =>
  (retract ?item)
  (printout t t t "The power switch is on the right side of PC to the back." t

            " The O setting means OFF and the 1 setting means ON." t
            " If the Power is off (O)" t
            "   Make sure the electrical outlet is working." f
            "   Make sure the power cables are connected." t
            "   Make that all fuses are in good condition." t
            "   Turn Power On " )
  (printout t t t "The fan should make a noise when power is on."
            " Make sure that video monitor is also on. " t
            " Check for any error massages that may appear." t )
  (printout t t "If there is not a DOS boot disk in the disk drive," t
            " or the primary disk drive doesn't work," t
            " the IBM PC will start up in basic mode," t
            " Form basic mode you can execute basic language commands." t )
  (printout t t t "Did the power come on?" t
            "  Enter Y(yes), N(no), D(don't know) : " )
  (assert (power on =(read)) ))
```

Figure 2.

This is a simple rule to tell the user to be sure that the power is on, but one that our inexperienced repair persons suggested as an addition after prototype testing.

This phase has two parts: (1) simple checks to be made with the cover on the machine and (2) simple checks to be made with the cover off the machine. Some checks that the system prompts the user to perform with the cover on are:

(1) check all cables
(2) check power
(3) check monitor power and adjustments
(4) check that software loaded correctly and drive doors closed
(5) reboot system

In addition to the above checks, the users are given as much basic information about the components of a personal computer as possible. For example, one help screen that can be invoked by the rules helps the user identify which card is the graphics adapter card. Some of the checks the user is prompted to perform once the cover is removed are:

(1) check the dip switches
(2) check to see if all boards tight
(3) check to see if motherboard chips are tight
(4) check to see if all internal cables connected are tight
(5) check dip switches on component boards
(6) reboot

Considerable extra information is given the user about the safe procedures to use when working on the computer, such as when to shut off the power to protect a board before it is reseated.

CHECK OF ERROR CODES

The error codes that can be generated by a personal computer during its boot procedure can be massive. While lists are available, they are often missing at repair-time. We included a set of rules in our expert system that would analyze the error code produced by the computer and suggest which components were most likely to be defective. Some of the most popular component boards also have error codes and we added modules for these as well. One of the most useful of the error codes is a memory bank error. One of the rules for our memory bank error check is given in Figure 3.

```
(defrule memory-error
   ?x <- (phase memory-bank-check y)
   =>
   (retract ?x)
   (system "cls")
   (printout t "If an error code is displayed in the pattern of 'XXXX 201 '")
   (printout t crlf "where X is any number or letter and:          " crlf)
   (printout t crlf)
   (printout t "If the first X is '0', there is a memory problem on ")
   (printout t "the system board.                                  ")
   (printout t crlf "If the second X is '0', failure in bank 0   ")
   (printout t crlf "If the second X is '4', failure in bank 1   ")
   (printout t crlf "If the second X is '8', failure in bank 2   ")
   (printout t crlf "If the second X is 'C', failure in bank 3   ")
   (printout t crlf)
   (printout t crlf "If the last two XXs are:                    ")
   (printout t crlf " '00'          replace parity module of the bank")
   (printout t crlf " '01'          replace module '0' of the bank")
   (printout t crlf " '02'          replace module '1' of the bank")
   (printout t crlf " '04'          replace module '2' of the bank")
   (printout t crlf " '08'          replace module '3' of the bank")
   (printout t crlf " '10'          replace module '4' of the bank")
   (printout t crlf " '20'          replace module '5' of the bank")
   (printout t crlf " '40'          replace module '6' of the bank")
   (printout t crlf " '80'          replace module '7' of the bank" crlf)
   (printout t crlf " Are you still having problems? (y or n):")
   (assert (answer memory-bank-check =(read))))
```

Figure 3

In general, the error codes for our 8088 expert system were good enough so that this module could tell the user exactly what board to replace. If there were no error codes or if this module could not decide from the error codes what the problem was, the user would go to the next module.

COMPONENT CHECKS

This phase of the expert system begins by asking the user to reboot the system after a 30 second delay and then to pay careful attention to the questions asked in Figure 4.

```
(1)  Is the fan running (y or n).  Listen closely for the
     fan.  It may be hard to hear.
(2)  The cursor appears on the monitor screen (y or n).
(3)  You hear one beep (as opposed to no beep or multiple
     beeps) (y or n).
(4)  The light on the disk drive comes on and you hear the
     drive motor running inside the disk drive (y or n).
(5)  The screen comes up in BASIC if not booting from DOS (y
     or n).
```

Figure 4.

This menu is used to select one of the following modules of rules for the user.

(1) power supply
(2) graphics adapter and monitor
(3) keyboard
(4) floppy disk controller and drives

These modules were seen as those most often giving trouble at this point by our domain experts. If the user's answers to the first menu questions result in no rules in (1) to (4) being used, then a number of other options are presented to the user each of which can trigger a module if the user answers the prompting questions with enough information to indicate that a diagnostic module should be invoked. The main option that we have implemented to date is a hard drive and adapter module. Options to check communications cards and printer cards have also been added. In future versions we would like to add modules to cover modems and tape drives. There are many interrelated rules in this phase. A typical section of code from our current system is shown in Figure 5.

```
(defrule No-motor
     ?x <- (motor-spinning n)
     ?y <- (phase disk drive)
     =>
     (retract ?x ?y)
     (printout t crlf "Please check the power supply unit." crlf)
     (printout t "If the power supply is ok, then replace the disk drive.")
     (printout t crlf)
     (assert (last-choice 3)))

(defrule check-BASIC
     (phase BASIC)
     =>
     (system "cls")
     (printout t crlf "Insert a DOS disk in drive A: and reboot." crlf)
     (printout t crlf "Watch for DOS to boot up." crlf)
     (printout t crlf "Did DOS boot? (y or n): ")
     (assert (DOW BOOTS =(read))))
```

Figure 5.

SUMMARY

Our current system has about 150 rules. This is a little misleading since some of the rules result in the user being presented a menu selection that in reality is really another set of rules. The system works fairly well, although we consider it still to be in the development phase. For many problems, the user will be guided to exactly what's wrong. In some cases the system gives up because it does not have enough knowledge. It is clear that such a system could be perfected. What is not as clear is whether or not it is really practical to develop such a system for every type of personal computer. We think it is and plan to continue our development of the basic system.

Some prototypes of our current systems have been developed as class projects in our Expert Systems course at The University of Texas at Tyler over the past few years and several students have developed prototype systems as independent study projects. The final system described in this paper has had input from many students at our University. Unfortunately, there were too many to give credit to all by name, thus, I simply mention their contribution in this summary.

As an extension of the ideas of this paper we are now applying the basic systems architecture to developing a hardware diagnostic system for any system with many individual components. As a part of our extension we are embedding several neural networks that will determine when sets of electrical and mechanical signals should fire a CLIPS rule. At this point it appears that our basic architecture is a solid basis for this extension.

REFERENCES

IBM Technical Reference for the IBM PC and Options and Adapters.
IBM Hardware Reference for the IBM PC.
Giarrantano and Riley. (1989). Expert Systems, PWS-KENT.
Hayes-Roth, Waterman, Lenat. (1983). *Building Expert Systems,* Addison-Wesley.
Michalski, Carbonel and Mitchell. *(1986. Machine Learning: An Artificial Intelligence Approach-Vol. 2,* Morgan Kaufmann.
Westphal, Chris (ed). (1989). *SIGART Newsletter (special edition on Knowledge Acquisition),* ACM Publications, New York, N.Y. 10249, pp 6,198.
Whitson, Wu and Taylor. (1990). *Using an Artificial Neural System to Determine the Knowledge Base of an Expert System,* Proceedings of the ACM Symposium on Small Computers, Washington.